I CLAIM:

1. A method of managing execution time in a shared memory parallel processor computing environment in which a plurality of independent processors execute processes simultaneously, comprising steps of:

defining a plurality of process classes and assigning each process to be executed to a one of the process classes;

defining an execution time slice for each of the process classes; and

permitting a process to be executed by a one of the processors without interruption until the execution time slice associated with the process class has expired.

2. A method as claimed in claim 1 further comprising a step of enabling processes of at least one of the process classes to call a lock procedure during execution, which permits the process to continue to be executed without interrupt for a predefined period of time after the time slice associated with the process class has expired.

3. A method as claimed in claim 2 further comprising a step of enabling the at least one of the processes to call an unlock procedure to permit execution of the process to be terminated before the predefined period of time has expired.

[redacted]

4. A method as claimed in claim 2 wherein the predefined period of time is an integer multiple of the time slice associated with the process class.

5. A method as claimed in claim 2 wherein the lock procedure may be called repeatedly by the same process.

6. A method as claimed in claim 5 further comprising a step of keeping a count of lock states declared by the lock procedure.

7. A method as claimed in claim 6 further comprising a step of decrementing the count each time the unlock procedure is called.

8. A method as claimed in claim 7 further comprising a step of testing the count each time the count is decremented to determine if the count is greater than zero, and declaring the process unlocked if the count is not grater than zero.

9. A method as claimed in claim 8 wherein declaring a process locked and declaring a process unlocked comprises a step of setting a lock flag to a first value when the process is declared locked and setting the lock flag to a second value when the process is declared unlocked.

10. A method as claimed in claim 9 wherein the lock flag is a Boolean variable.

11. A method as claimed in claim 8 further comprising a step of terminating execution of the process if the process is declared unlocked after the time slice allocated to the process has expired.

12. A method as claimed in claim 8 further comprising a step of permitting the process to continue execution for a remainder of the time slice if the process is declared unlocked before the time slice allocated to the process has expired.

13. A method as claimed in claim 1 wherein the execution time slice is stored in a time slice counter maintained by a processor that executes the process.

14. A method as claimed in claim 13 wherein the time slice counter is initialized by a scheduler that schedules the process to be executed by the processor.

15. A method of managing a timer queue in a shared memory parallel processor computing environment in which a plurality of independent processors execute processes simultaneously, the timer queue being used to queue processes in a wait state until a predetermined process removal time has expired, the method comprising steps of:

defining a variable for storing a time at which a next process is to be removed from the timer queue;

periodically examining the variable without generating a system interrupt to determine

whether the time stored in the variable is less than or equal to an instant system time;

removing each process from the timer queue which has an associated removal time that is less than or equal to the instant system time; and

re-computing a time at which a next process is to be removed from the timer queue, and storing the re-computed time in the variable.

16. The method as claimed in claim 15 wherein adding new processes to the timer queue further comprises steps of:

computing a delay time specified by the process to be added to the queue;

comparing the computed delay time with the time stored in the variable;

if the computed delay time is less than the delay time stored in the variable, replacing the delay time stored in the variable; and

adding the process to the timer queue.

17. The method as claimed in claim 15 further comprising a step of re-ordering the timer queue each time at least one process is removed.

18. The method as claimed in claim 15 further comprising a step of placing processes removed from the timer queue into a ready queue for execution by a one of the processors.

19. The method as claimed in claim 18 wherein the ready queue in which a process is placed is governed by a class with which the process is associated.

20. The method as claimed in claim 19 wherein the ready queue is a first-in-first-out (FIFO) queue and the process is placed at a rear end of the FIFO queue.

21. A shared memory parallel processor computing machine in which a plurality of independent processors simultaneously execute processes, comprising:

means for associating each process to be executed with a process class that defines rights and priorities associated with the process;

means for associating an execution time slice with each of the process classes; and

means for monitoring a process during execution to permit the process to be executed by a one of the processors without interruption until the execution time slice associated with the process class has expired.

22. A computing machine as claimed in claim 21 wherein the means for permitting a process to be executed without interruption comprises a time slice counter that is initialized to a count representative of the predetermined execution time slice when the process is scheduled to be run, and is decremented at predetermined time intervals while the process is being executed.

23. A computing machine as claimed in claim 22 wherein the time slice counter is initialized by a scheduler program that schedules the process to be executed by the processor.

24. A computing machine as claimed in claim 22 wherein the time slice counter generates a hardware interrupt when the time slice is decremented to zero.

25. A computing machine as claimed in claim 21 further comprising a lock procedure that may be called by predetermined processes to permit the processes to continue to execute after the execution time slice has expired.

26. A computing machine as claimed in claim 25 wherein the predetermined processes are defined by a class attribute that indicates that processes belonging to an associated class can call the lock procedure.

27. A computing machine as claimed in claim 25 wherein the lock procedure sets a lock flag to indicate that the process is locked.

28. A computing machine as claimed in claim 25 wherein the lock procedure also initializes a lock time counter to a predetermined value.

29. A computing machine as claimed in claim 28 wherein the lock procedure initializes the lock counter to an integer multiple of the process time slice.

30. A computing machine as claimed in claim 25 further comprising an unlock procedure that may be called by the predefined processes enabled to call the lock procedure.

31. A computing machine as claimed in claim 30 wherein the predefined procedures are permitted to call the lock procedure more than once in succession without calling the unlock procedure.

32. A computing machine as claimed in claim 21 further comprising a timer queue for queuing processes to be executed for a predetermined period of time, the timer queue being managed without interrupt until a process in the timer queue is ready to be transferred to a ready queue for execution by a one of the processors.

33. A computing machine as claimed in claim 32 further comprising a timer queue variable for storing a time at which a next process is to be removed from the timer queue and placed in the ready queue.

34. A computing machine as claimed in claim 33 further comprising a scheduler for periodically comparing the timer queue variable with a current system time to determine when a next process is to be removed from the timer queue and placed in the ready queue.

35. A computing machine as claimed in claim 34 wherein the scheduler is further adapted to compute a time at which a process is to be removed from the timer queue when the process is added to the timer queue, and the

scheduler is further adapted to update the timer queue variable if the timer queue variable is greater than the time computed by the scheduler.

36. A computing machine as claimed in claim 35 wherein the scheduler program is further adapted to remove at least one process from the timer queue each time the timer queue variable is less than or equal to a current system time, and to reinitialize the timer queue variable based on a time at which an next process is to be removed from the timer queue.